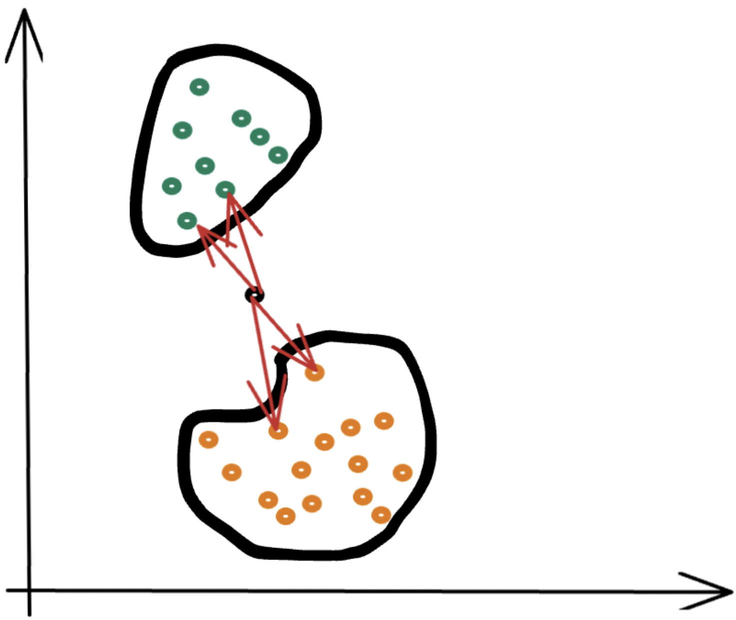
第三章 KNN

3.1 基本介绍

k近邻法（K-nearest neighbor，knn）是一种基本分类与回归方法。简单、直观来说，其在给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的K个实例，这个k个实例的多数属于某个类就把该输入的实例分为这个类。因此，k近邻法不具有显式的学习过程。



3.2 k近邻模型

k近邻法使用的模型实际上对应于特征空间的划分。模型由三个基本要素——距离度量、k值选择和分类决策规则决定。

3.2.1 距离度量

L_p距离或$Minkowski$距离定义如下：

$$L_p(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^p)^{\frac{1}{p}}$$

特别得，当$p = 1$时，该距离称为曼哈顿距离；当$p = 2$时，称为欧式距离。在二维下，$x_1(2, 4)$与$x_2(5, 7)$之间的欧式距离计算式如下：

$$L_2(x_1, x_2) = \sqrt{(2-5)^2 + (4-7)^2}$$

3.2.2 k值选择

k值的选择会对k近邻法的结果产生重大影响。k值的减小意味着整体模型变得复杂，容易发生过拟合。如果k值取得较大，意味模型变得简单，使得预测容易发生错误。在实际应用中，k值一般取一个比较小的数值，且一般取奇数。同时，k也往往作为超参数搜寻得对象。k值的选择反映了对近似误差和估计误差之间的权衡，通常由交叉验证选择最优的k。

3.2.3 分类决策

一般采用多数表决规则。如果$N_k(x)$的区域类别是c_j，那么误分类的概率是

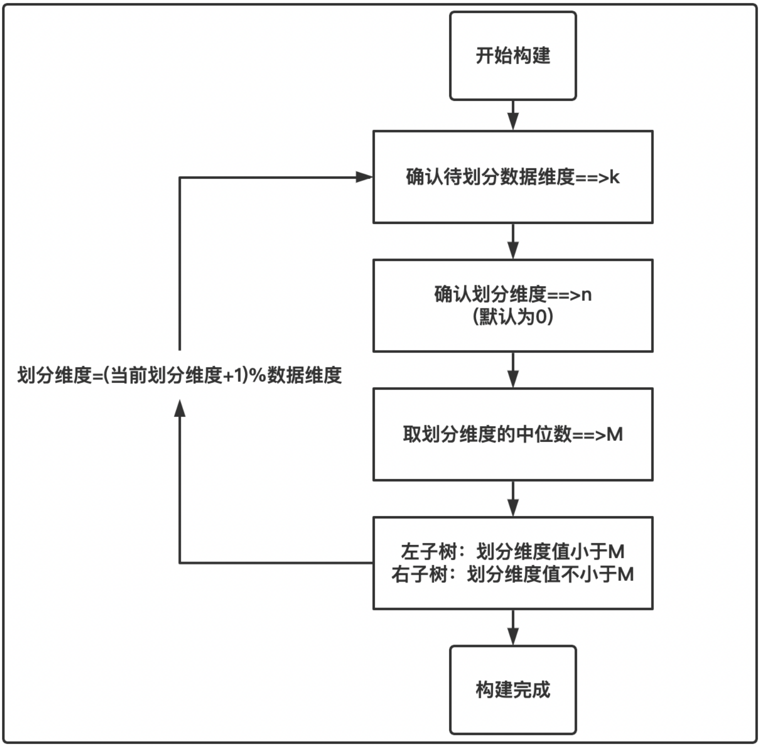
$$\frac{1}{k}\sum_{x_i\in N_{k(x)}} I(y_i\neq c_j)=1-\frac{1}{k}\sum_{x_i\in N_k(x)} I(y_i=c_j)$$

所以，要使误分类率最小即经验风险最小，就要使 $\sum_{x_i\in N_{k(x)}} I(y_i=c_j)$最大。所以多数表决规则等价于经验风险最小化。当然除了多数表决这种规则，还要距离同等权重，以及距离越近，权重越大。

3.3 构造kd树

k近邻法最简单的实现方法是线形扫描，但是当训练数据集非常大的时候，这种方法是不可行的。为了提高k近邻搜索效率，可以考虑使用特殊的结构储存训练数据，以减少计算距离的次数。使用kd树就是一种有效的方法【注意这里的kd树是储存k维空间数据的树结构，这里的k与k近邻法中的k意义不同】，构造平衡kd树具体算法如下：

> **输入**：k维空间数据集T={$x_1, x_2, \cdots, x_N$}，其中 $x_i=(x_i^{(1)}, x_i^{(2)}, \cdots, x_i^{(k)})^T$,$i=1,2,\cdots$，N；
> **输出**：kd树

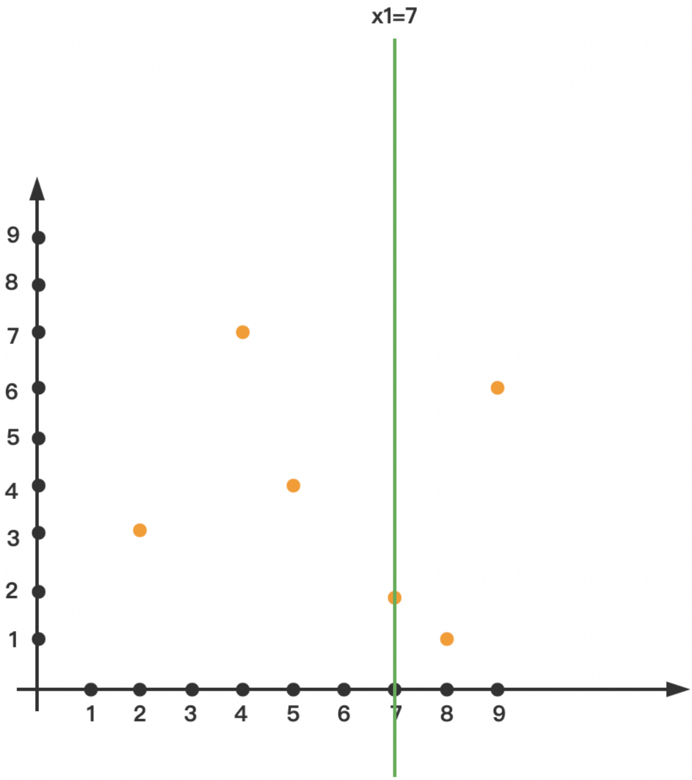


例：给定一个二维空间数据集：

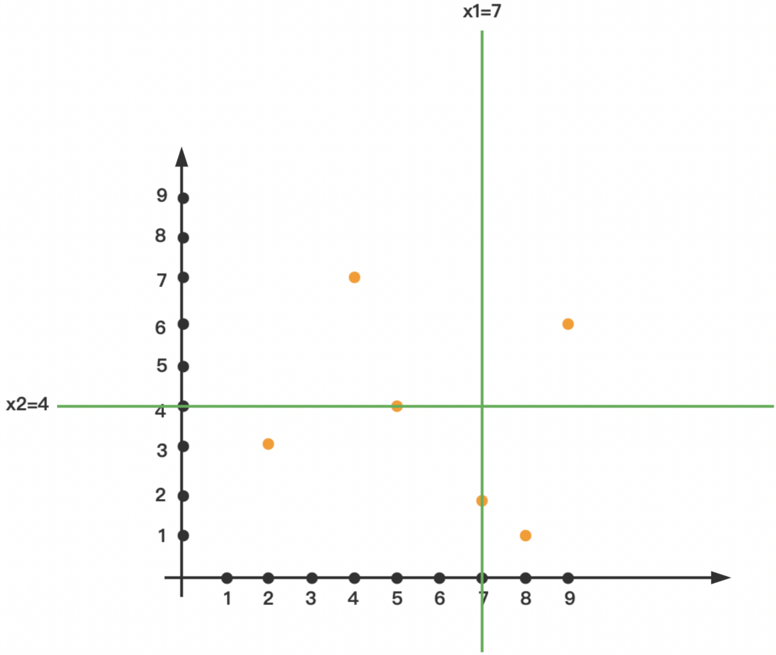
$$T=\{(2,3)^T,(5,4)^T,(9,6)^T,(4,7)^T,(8,1)^T,(7,2)^T\}$$

构造一个kd平衡树。

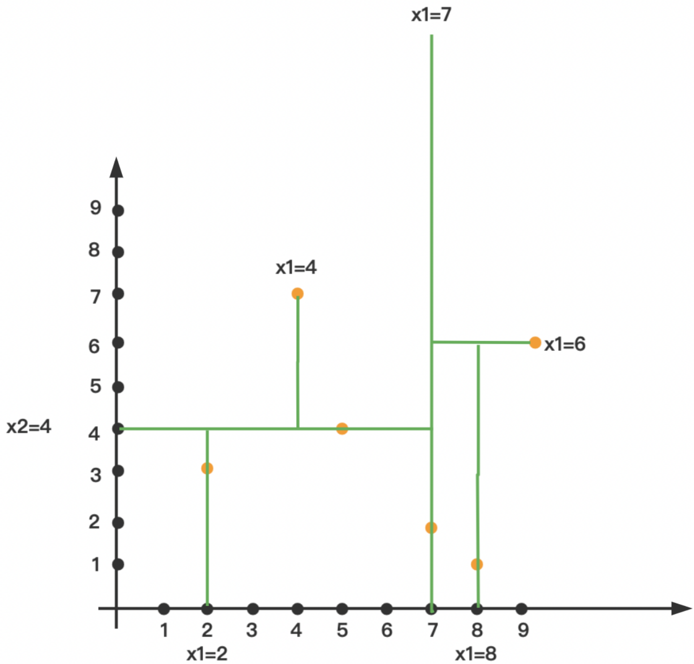
解：观察$x^{(1)}$中6个点值分别为2，4，5，7，8，9=======>中位数点为7，以平面$x^{(1)}=7$将空间分为左、右两个子矩形，此事根节点为$(7,2)$；如下图所示：



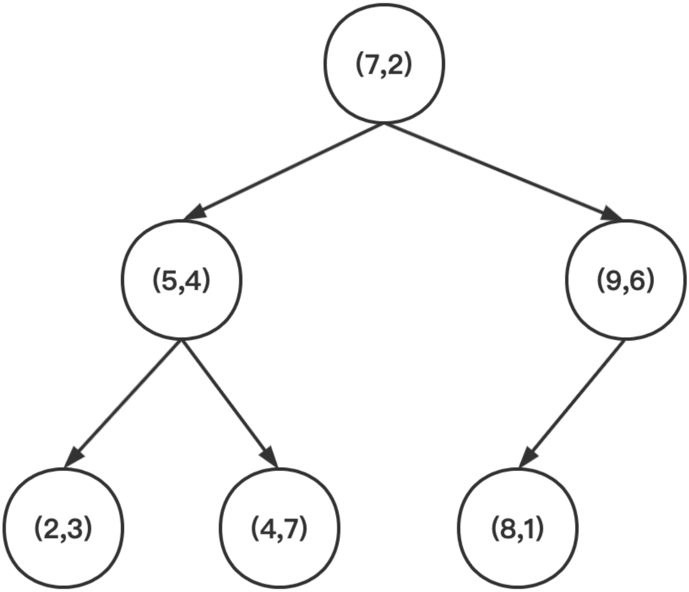
接着，左矩形中，以$x^{(2)}=3$，4，7，以$x^{(2)}=4$分为两个矩形，此时左边第一节点为$(5,4)$如下所示：



再在上下矩形进行拆分（大在右边，小在左边），针对 $x^{(1)}=7$的右边同理，最终拆分结果如下图所示（注意：在三维中，这些线条就是切割平面）：



按照刚才划分的顺序，构造的平衡kd树如下：



3.4 搜索kd树

3.4.1 算法

> 输入：已构造的kd树，目标x
> 输出：x的最近邻点
> (1) 寻找“当前最近邻点”
>
> - 从根节点出发，递归访问kd树，找出包含x的叶节点；
> - 以此叶节点为“当前最近点”

> (2) 回溯
>
> - 若该节点比“当前最近点”距离目标点更近，更新“当前最近点”；

> - 当前最近点一定存在于该节点一个子节点对应的区域，检查子结点的父结点的另一子节点对应的区域是否有更近的点

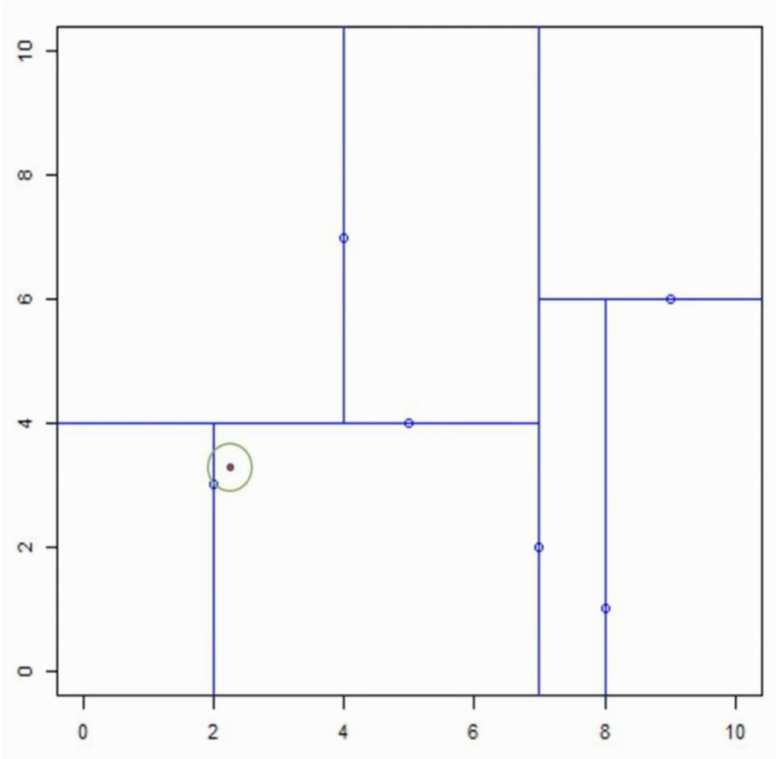
> (3) 当回退到根节点时，搜索结束，最后的“当前最近点”即为x的最近邻点。

3.4.2 寻找最近点

基于上面的算法可能仍然比较混乱，这时候，我们在3.3的基础之上来具体演示这个算法是如何工作的。

3.4.2.1 案例1

- 在上面3.3的例子中，找到$x=(2.1,3.1)$的最近点。从根节点出发，递归地向下访问kd树。若目标点x当前维的坐标小于且分点的坐标，则移动左节点，否则，移动到右节点。直到子节点为叶节点为止。【此时，最近点为(2,3)】

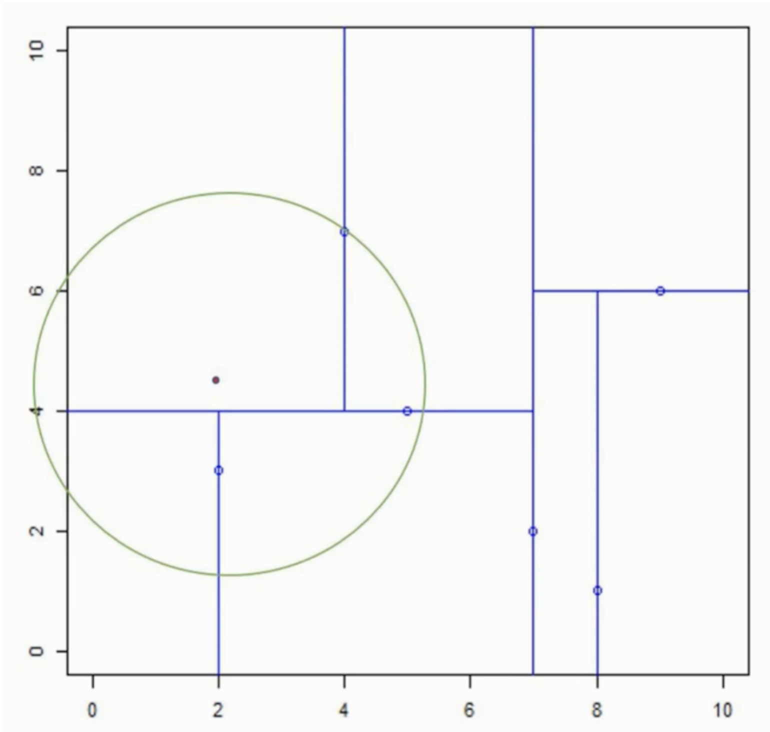


- 以我们的目标点为圆心，到该矩形中最近点的距离作为半径r，如图中绿圈所示；
- 开始回溯：
  - 因为该绿圈没有与其父节点的切割平面$x^{(2)}=4$相交，所以在$x^{(2)}=4$的上部分区域没有比(2,3)离目标点更近的点。
  - 再顺着节点往上走，其与父节点的父节点的切割平面$x^{(1)}=7$也没有相交，说明在$x^{(1)}=7$的右边矩形区域中也没有比(2,3)离目标点更近的点了。
  - 综上,(2,3)就是距离目标最近的点。

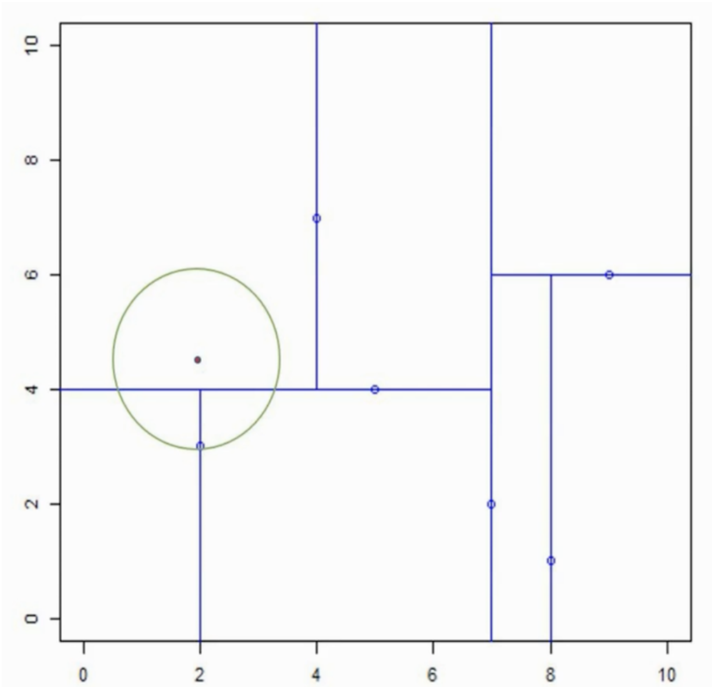
3.4.2.2 案例2

案例1中的例子似乎有些特别，在这里，我们用上面相同的办法来寻找$x=(2,4.5)$的最近点。

- 同样，先根据二叉树的特点（左小右大、下小上大），确定目标点所在的位置，并在此矩形中找到距离目标点最近的点，并以它们之间的距离作为半径，目标点作为圆心画圆。如图中绿色的圆所示。



- 开始回溯：
  - 因为该圆与目前最近点的父节点的切割超平面$x^{(2)}=4$相交，因此，我们要检查该父节点的另一子节点对应的区域是否有更近的点。显然是有的，即$x=(2,3)$是比$x=(4,7)$距离目标更近的点。
  - 更新最近点$x=(2,3)$，并根据上面的原则，重新绘制这个圆，如下图所示。



- 这时，我们还是按照上述的规则，检查该圆与最近点的父节点是否有相交，发现是相交的。但是该父节点的另一子节点区域没有比该最近点到目标点更近的点，因此再检查该圆形是否与父节点的父节点的切割超平面相交，这里是$x^{(1)} = 7$，没有相交，且$(7, 2)$为根节点。最终确定了目标点$(2, 4.5)$的最近点为$(2, 3)$

4、sklearn API

[最近邻]()